CSE 451: Operating Systems
Winter 2004

Module 15
BSD UNIX Fast File System

Ed Lazowska
lazowska@cs.washington.edu
Allen Center 570

Advanced file system implementations

- We've looked at disks
- We've looked at file systems generically
- We've looked in detail at the implementation of the original Bell Labs UNIX file system
  - a great *simple yet practical* design
  - exemplifies engineering tradeoffs that are pervasive in system design
- Now we'll look at two more advanced file systems
  - Berkeley Software Distribution (BSD) UNIX Fast File System (FFS)
    - enhanced performance for the UNIX file system
    - at the heart of most UNIX file systems today
  - Berkeley Log-Structured File System (LFS)
    - a research file system – a worth experiment

BSD UNIX FFS

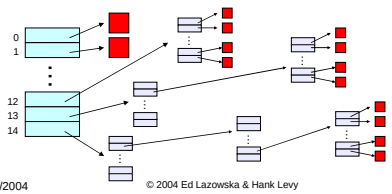
- Original (1970) UNIX file system was elegant but slow
  - poor disk throughput
    - far too many seeks, on average
- Berkeley UNIX project did a redesign in the mid '80's
  - McKusick, Joy, Fabry, and Leffler
  - improved disk throughput, decreased average request response time
  - principal idea is that FFS is aware of disk structure
    - i.e., place related things on nearby cylinders to reduce seeks

Recall the UNIX disk layout

- Boot block
  - can boot the system by loading from this block
- Superblock
  - specifies boundaries of next 3 areas, and contains head of freelists of inodes and file blocks
- i-node area
  - contains descriptors (i-nodes) for each file on the disk; all i-nodes are the same size; head of freelist is in the superblock
- File contents area
  - fixed-size blocks; head of freelist is in the superblock
- Swap area
  - holds processes that have been swapped out of memory

Recall the UNIX block list / file content structure

- directory entries point to i-nodes – file headers
- each i-node contains a bunch of stuff including 15 block pointers
  - first 12 point to file blocks (i.e., 4KB blocks of file data)
  - then single, double, and triple indirect indexes



UNIX FS data and i-node placement

- Original UNIX FS had two major performance problems:
  - data blocks are allocated randomly in aging file systems
    - blocks for the same file allocated sequentially when FS is new
    - as FS "ages" and fills, need to allocate blocks freed up when other files are deleted
      - deleted files are essentially randomly placed
      - so, blocks for new files become scattered across the disk!
  - i-nodes are allocated far from blocks
    - all i-nodes at beginning of disk, far from data
    - traversing file name paths, manipulating files, directories requires going back and forth from i-nodes to data blocks
- BOTH of these generate many long seeks!

FFS: Cylinder groups

- FFS addressed these problems using the notion of a cylinder group
  - disk is partitioned into groups of cylinders
  - data blocks from a file are all placed in the same cylinder group
  - files in same directory are placed in the same cylinder group
  - i-node for file placed in same cylinder group as file's data
- Introduces a free space requirement
  - to be able to allocate according to cylinder group, the disk must have free space scattered across all cylinders
  - in FFS, 10% of the disk is reserved just for this purpose!
    - good insight: keep disk partially free at all times!
    - this is why it may be possible for df to report >100% full!

FFS: Increased block size, fragments

- I lied: the original UNIX FS had 1KB blocks, not 4KB blocks
  - even more seeking
  - small maximum file size (¼ as much user data per block, ¼ as many pointers per indirect block), ~17GB maximum file size
- FFS fixed this by using a larger block (4KB)
  - allows for very large files (4TB)
  - but, introduces internal fragmentation
    - on average, each file wastes 2K!
      - why?
    - worse, the average file size is only about 1K!
      - why?
  - fix: introduce "fragments"
    - 1KB pieces of a block

FFS: Awareness of hardware characteristics

- Original UNIX FS was unaware of disk parameters
- FFS parameterizes the FS according to disk and CPU characteristics
  - e.g., account for CPU interrupt and processing time, plus disk characteristics, in deciding where to lay out sequential blocks of a file, to reduce rotational latency

FFS: Faster, but less elegant (or, "as ugly as the day is long"!)

- Multiple cylinder groups
  - effectively, treat a single big disk as multiple small disks
  - additional free space requirement (this is cheap, though)
- Bigger blocks
  - but fragments, to avoid excessive fragmentation
- Aware of hardware characteristics
  - ugh!